ANL/ALCF/ESP-17/3

Argonne NATIONAL LABORATORY

Large-Scale Simulation of Brain Tissue, Blue Brain Project, EPFL

Technical Report for the ALCF Theta Early Science Program

Argonne Leadership Computing Facility

ALCF Early Science Program (ESP) Technical Report

ESP Technical Reports describe the code development, porting, and optimization done in preparing an ESP project's application code(s) for the next generation ALCF computer system. This report is for a project in the Theta ESP, preparing for the ALCF Theta computer system.

ANL/ALCF/ESP-17/3

Large-Scale Simulation of Brain Tissue, Blue Brain Project, EPFL

Technical Report for the ALCF Theta Early Science Program

edited by
Timothy J. Williams and Ramesh Balakrishnan

Argonne Leadership Computing Facility

prepared by
Fabien Delalondre, Felix Schuermann, Eilif Muller, and Marc Oliver Gewaltig

May 2018

This work was supported in part by the Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

Large-Scale Simulation of Brain Tissue, Blue Brain Project, EPFL

Fabien Delalondre*[1], Felix Schuermann†[1], Eilif Muller‡[1], and Marc Oliver Gewaltig §[1]

[1] Ecole Federale Polytechnique de Lausanne, Blue Brain Project

1 Science Summary

The initial version of this project described in the awarded proposal aimed at largely focusing on the evaluation of Intel KNL hardware and the development of new scientific use cases such as the neurorobotics and whole brain modeling. However, during the course of the project EPFL has been awarded a high impact INCITE project focused on the development of models to support brain plasticity which is considered as the key toward understanding memory. Considering this very prestigious opportunity a large part of our developments has been refocused on the development of methods to support the plasticity use case which is described in Annex 1 along with the awarded INCITE proposal. [*Proposal not included here. -Ed.*]

2 Codes, Methods and Algorithms

NEURON software has been developed over the last 30 years by Michael Hines from Yale University to support the electrical simulation of large scale neural networks where neurons can be modeled as point neurons or morphologically detailed neurons. The software is implemented in C and C++, Python and HOC interpreted languages. To maximize users productivity, NEURON also includes support for the NMODL language which implements a DSL specific to neuroscience. At compilation time, NMODL files are translated into C code which is then compiled to build the NEURON executable. NEURON software distributed implementation is using MPI and more recently OpenMP and proved to scale up to 32k processors of IBM Blue Gene/L. The code is mostly memory capacity bound and memory bandwidth bound. It requires the resolution of a very large number of small kernels representing synapses (memory bandwidth bound) and ion channels (either compute or memory bandwidth bound). However, and due to the need for NEURON to support a large set of use cases, NEURON implementation lacked a number of optimizations such as its memory usage leading to very large memory footprint requirements. coreNeurons effort started to resolve this issue and allow for explorations of new solutions. NEURON and coreNeuron now work together as described in Figure 1: NEURON and its high level proprietary layer Neurodamus

*fabien.delalondre@epfl.ch
†Felix.Schuermann@epfl.ch
‡Eilif.Mueller@epfl.ch
§Marc-Oliver.Gewaltig@epfl.ch

(implemented in HOC language) are used to setup the model which is built in memory by part with a footprint of about 18MB per neuron. The model is then dumped by chunks into files. coreNeuron memory footprint optimized implementation (About 3MB per neuron) loads the model specification of NEURON, assembles it in memory and starts the execution of the simulation.

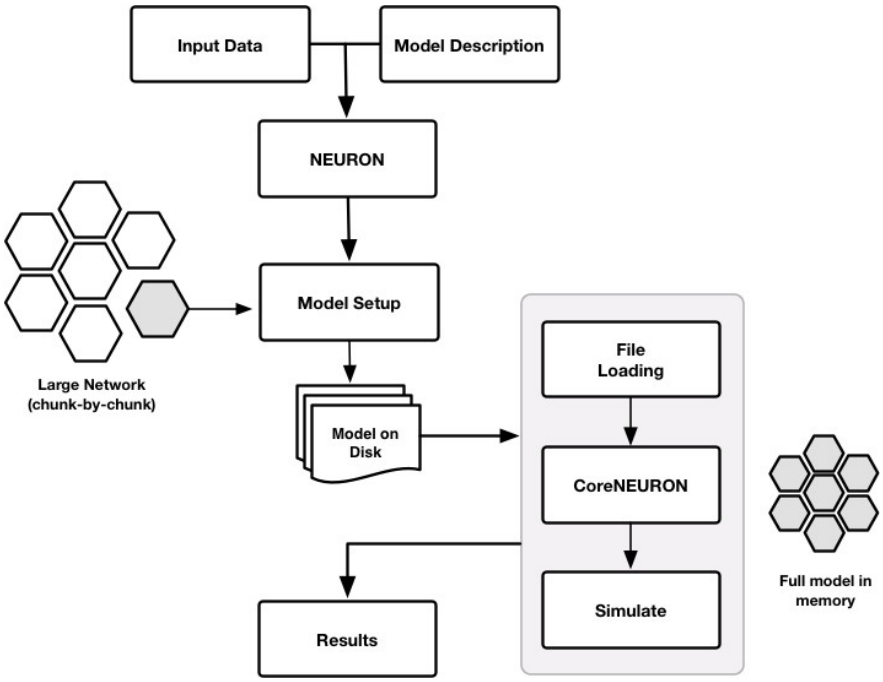

Figure 1: NEURON to coreNEURON workflow

3 Code Development & Refactoring

Due to the support of a large set of scientific projects, including an INCITE project, coreNeuron has gone through a large set of code developments as follows: At the beginning of the project, the workflow described in Figure 1 was relying on the manual execution of both NEURON and coreNEURON in sequence. To considerably improve the efficiency of the workflow and its scientific usability, a large amount of efforts has therefore been put on performing a large refactoring of coreNEURON to support the possibility to get it compiled as a library and be directly linked to NEURON so that NEURON high level layer Neurodamus could be used to drive the execution from the user point of view. This refactoring is now achieved and in use by our scientists. In addition, and in order to run a simulation using coreNEURON, the model must still be built using original Neurodamus and NEURON workflow. Since these consume substantially more memory, it is necessary to partition the circuit into smaller pieces and perform the model building one by one. This was originally done by manually creating subsets appropriately sized for the target compute partition and then selecting these subsets one at a time via an external bash script. This was a time- consuming task for users and prone to errors which could require regenerating all subsets. The latest version of Neurodamus now automatically partitions the circuit according to the available resources and in a single execution is able to build a subset, generate the model files, clear memory, and continue to the next piece. In the event that model building is interrupted before all subsets are completed, Neurodamus can resume after the last successfully generated subset.

Both of those developments brought considerable value to our users and contributed to make coreNeuron eventually usable and used by our community as opposed to being considered as an optimized version of NEURON only HPC experts could use at large scale.

3.1 Configurations and Optimization for Theta

Thanks to prior work done in the European Project Dynamic Exascale Entry Platform (DEEP), NEURON/coreNeuron software was already made ready for Intel KNC architecture and kernels optimized for this type of architecture [6]. Therefore, a large part of the time has first been dedicated to studying Intel KNL architecture and Theta configuration to understand how to tune the setup of the software to eventually best leverage Thetas performance.

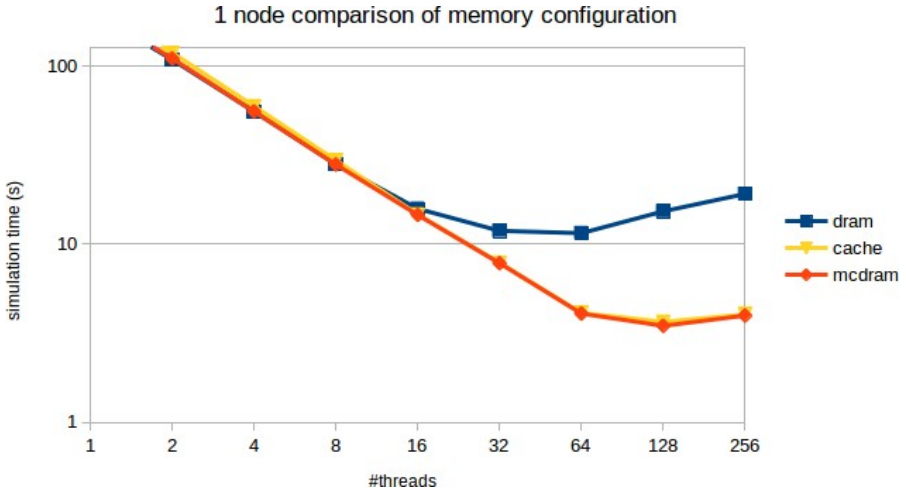

Figure 2: Single node performance of coreNeuron for various configurations of Intel KNL

We first used JLSE testbed system to test all available KNL configurations. Although most of available configurations have been tested, Figure 2 only shows the most two interesting one, that is when the KNL was booted in MCDRAM FLAT and Cache modes. From this Figure, we concluded that using KNL cache mode was bringing significantly good performance while considerably simplifying the implementation of the code. For the sake of completeness, we have also done a preliminarily implementation of some of the kernels using MCDRAM flat mode and identified using different tools a number of heavily used data structures to be first placed in the MCDRAM cache. However, we observed that such code division would require significant refactoring of the software while bringing a limited boost of performance. We therefore decided to focus more on the advancement of software to enhance community support in addition to working on preparing the next version of coreNEURON to target very large-scale simulation.

coreNEURON distributed version relies on a mix of MPI and OpenMP where a number of neurons are assigned to each thread along with a work stealing algorithm. In Figure 3, we have tested different OMP scheduling policies and concluded that the guided policy was the optimal to use (6% faster than the static one for a simulation of 100ms).

Figure 4 shows an analysis of the simulation time of coreNEURON for various combinations of OMP threads per MPI task up to a total of 2048 cores. From this study we can see that the best result is obtained when using 4 OMP threads and 32 MPI ranks per node.

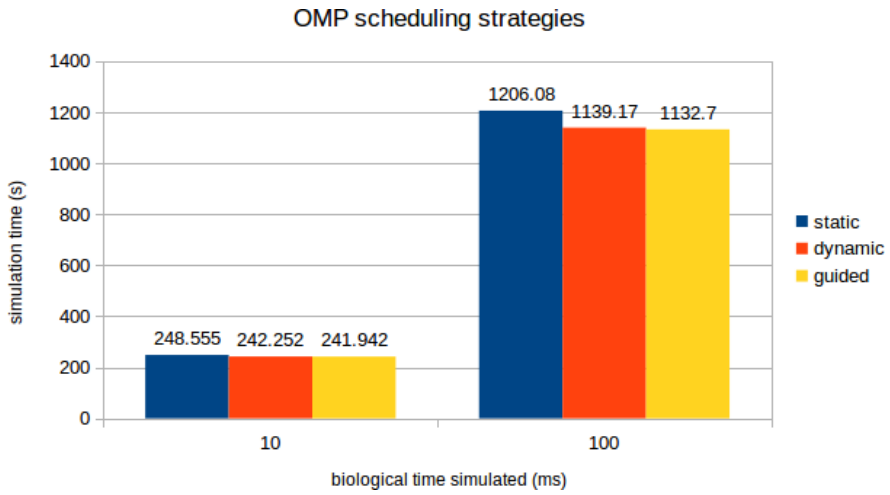

Figure 3: Analysis of the various OMP policy strategies.

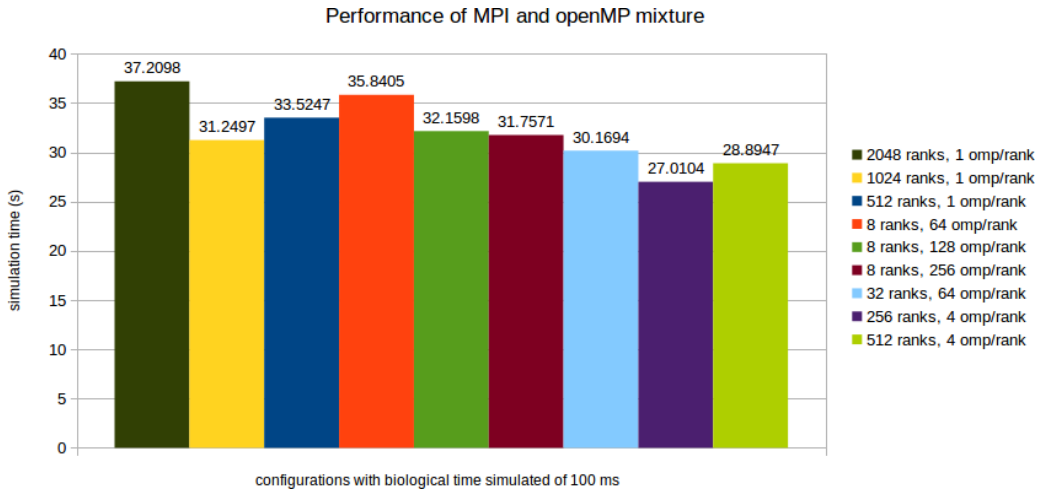

Figure 4: Comparison of various combinations of MPI and OMP setup.

Figure 5-a shows the resolution workflow of coreNEURON. All computation is independent of the other processes until it reaches the "spike_exchange" component which is responsible for supporting the exchange of spikes between the neurons. The frequency of exchange depends on what the biologist refers as the minimum spike delay which consists of the minimum amount of time at which neurons must be synchronized to avoid introducing too high of an error. In practice, the frequency of such an exchange corresponds to 4 or 5 time step resolutions. Figure 5-b shows a histogram of the execution time needed for the Spike Exchange routine (implemented using MPI_Allgather) responsible for communicating spikes between neurons. A well- balanced problem should exhibit very small variability between the different threads and make the Spike Exchange component only use about 2-6% of the total wall time (instead of taking more than 30% of the wall time when the problem is highly imbalanced). Even though it was not critical at this point, to further speed up this part of the code, it is worthwhile to point out that an alternative implementation referred as multisend (sending spike trains during multiple rounds through a minimum time delay [3]) has been migrated from NEURON to coreNEURON. In addition, several implementations of the queuing algorithm which is part of the Spike Exchange functionality have been investigated and summarized in [5] as part of the NeuroMapp library development.

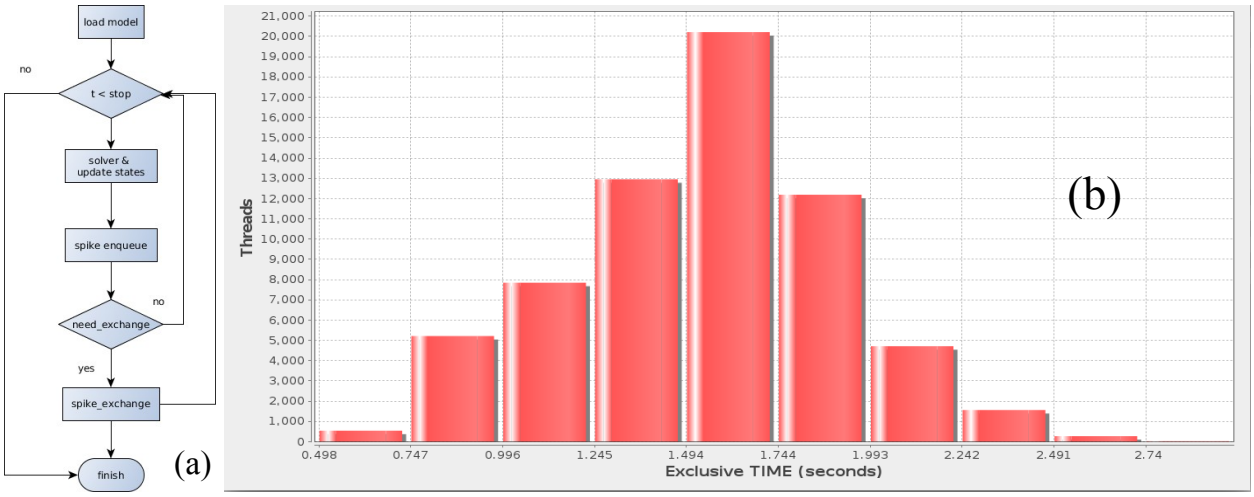

Figure 5: (a) coreNEURON resolution workflow and (b) Histogram of execution time of the Spike Exchange function for an execution with 65k processes. Well-balanced workload should show limited variations.

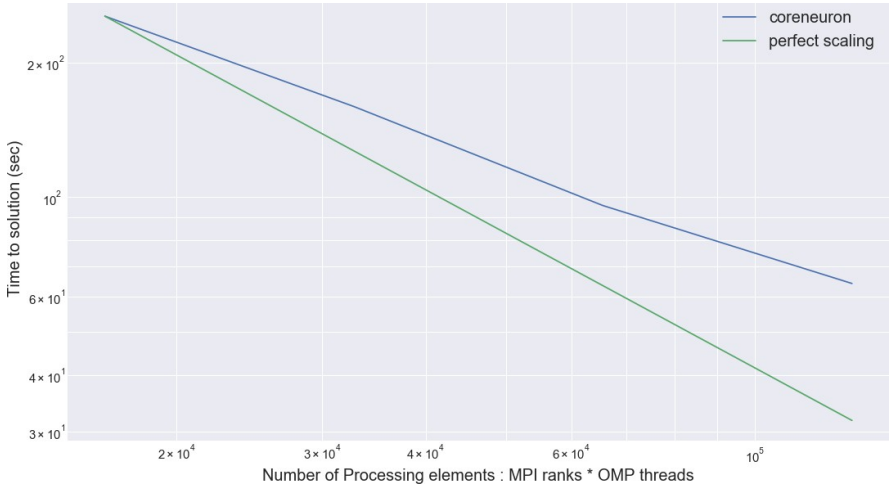

Figure 6: Strong scaling of coreNEURON up to 2k processors.

Using these optimal setups (KNL in cache mode, 32 MPI processes and 4 OMP threads per node, auto-vectorized and the used of SOA layout, OMP guided policy), we have performed a strong scaling study of coreNEURON on up to 2048 nodes of Theta (Figure 6). As BBP does not yet have a scientific use case which includes a large number of neurons to be included in the circuit, we had to artificially increase the size of the neural circuit to fit on 131k cores of Theta (2048 nodes). From this Figure, we can observe that even that coreNEURON loses scaling at high core counts although it proved to scale at more than 80% when there was enough load per processor. This result is due to the fact that the load per processor was still not high enough to get good results, demonstrating the need for a strategy that efficiently supports the subdivision of neurons into smaller parts.

3.2 New Methods/Algorithms

Load-balancing: coreNEURON strong scaling is currently limited by load-balancing: The simulation is synchronous, with a static work-load scheduling policy, pre- computed at the circuit creation in our previous simulation pipeline stage. The minimum partition size is equal to a whole neuron. Each neuron is different with different compute intensity and memory footprint due to cell morphology, electrical behavior, synapses and level of detail chosen for this simulation. Based on dry-runs time estimation performed with NEURON prior to the actual execution of the simulation, the static scheduler distributes using round robin sets of neurons to MPI ranks until an average set complexity is reached per MPI rank. As the response of the neurons is not localized to specific geographic locations of the neural network during the course of a simulation, such a workflow allows NEURON to achieve close to optimal balancing of the load across all MPI ranks. However, using the workflow described in Figure 1, coreNEURON used the results of the NEURON dry-run analysis to perform its own load balancing. As the two implementations come with different computing complexities, such a solution proved to be suboptimal. Thanks to the developments of coreNEURON as library we could fix this issue and have coreNEURON generating its own estimate and thus considerably fix its load balancing, reaching again the results obtained with NEURON.

The scheduling policy remains a limiting factor at very large scale as shown from ESP experiment: we are reducing the average number of cells per MPI rank, increasing load-imbalance. This limiting factor can be solved by considering other numerical methods that eventually permit to split a neuron into smaller computation tasks, leading to better scheduling opportunities such as the one below.

Supporting very strong scaling: To support the very strong scaling use case, NEURON software implements the multisplit algorithm [2] which supports neuron subdivision up to about 4-8 subdivision per neuron without loss of scaling. Beyond this threshold, the resolution of the very small linear tridiagonal algebraic system (of order equal to the number of neuron compartment which is about 300-400 per neurons) becomes the bottleneck for the scalability if one targets to subdivide neurons by a very large factor. As part of this project we have been looking at methods that could support subdividing neurons to the level of a single compartment to massively increase the level of parallelism of the simulation (targeting 100x order of magnitude more parallelism). We present here only preliminarily results of our implementation.

The stability of the explicit solver must respect the following condition

$$\Delta t < \frac{\Delta x^2}{2\alpha}$$
$$\alpha = \frac{1}{c_j r_j}$$

Where c_j and r_j correspond to the conductance and radius of the neuron j.

As introducing a new solver in coreNEURON or NEURON requires a lot of code refactoring, we decided to first use the NeuroMapp library [5], a miniapp library to evaluate the implementation of several solvers. As such, both implicit and explicit methods were introduced into NeuroMapp and comparison were made between the various solvers at the level of a single compartment. In Figure 7, we first validated the implementation of the explicit solver by verifying the onset of numerical instabilities when not respecting the conditions described by the stability equation.

In Figure 8, we have compared the results obtained by the implicit and explicit solvers and observed that both solvers were not exactly matching in terms of frequency but also in amplitude, leaving room for further numerical analysis. Further developments are currently carried out with the goal to further exploit the explicit solver and its degree of parallelism in order to possibly distribute the data only into L2 or L1 cache of the KNL.

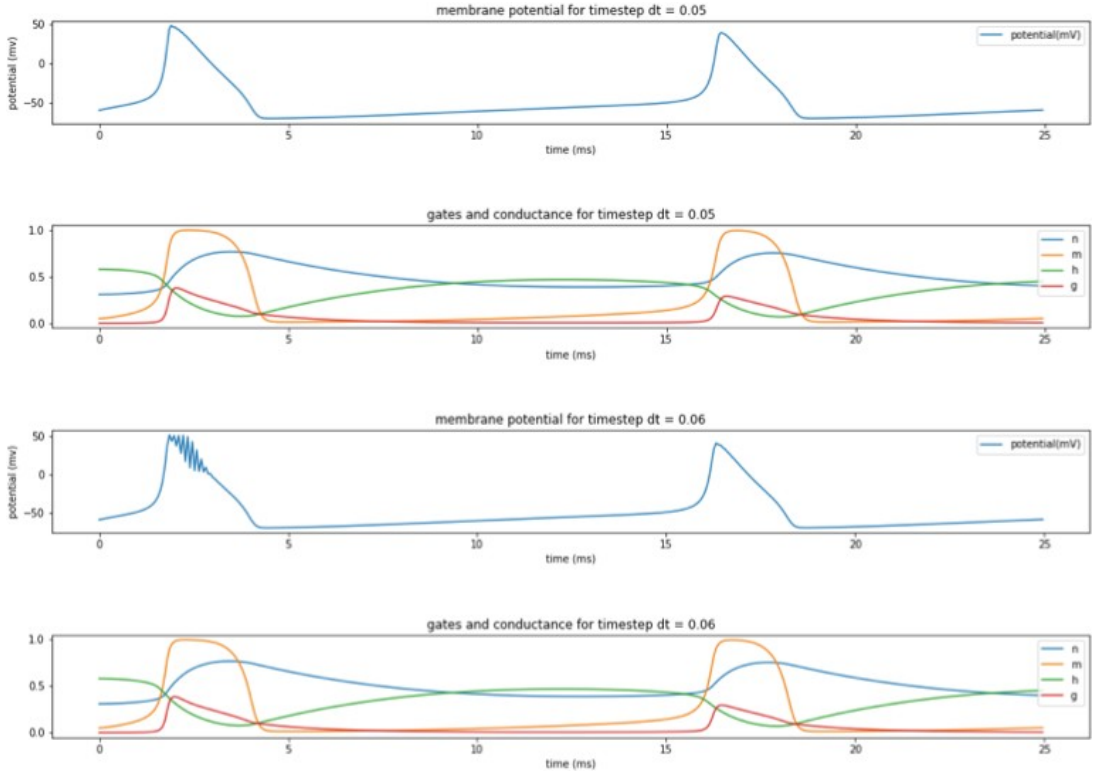

Figure 7: Validating the implementation of the explicit solver by verifying the onset of instabilities for the HH channel when using a Forward Euler solver. The evolution of the membrane potentials is provided for dt=0.05ms and dt=0.06ms (First and third rows) while the evolution of the channel gates is provided in both the second (dt=0.05ms) and fourth rows (dt=0.06ms).

Reducing memory footprint through in-memory compression: As one of the main challenges of the NEURON/coreNEURON application lies in its the large memory footprint we have investigated the possibility to support the compression of data in memory. Extensive details of the study are provided in [1] which is attached to this document in Annex 2 [*Not included here; see* `https://github.com/DevinBayly/gsoc_report/blob/master/report.pdf` *-Ed.*]. The conclusion of this preliminary study is that coreNEURON offers opportunities to compress the required data. However, it remains to be seen whether data compression will not eventually degrade the performance of the overall application. Further investigations must be conducted in the future.

4 Portability

Through its source to source compiler and the support of the NMODL, coreNEURON has been made portable across a number of architectures including x86, Intel KNL, NVIDIA GPU and IBM POWER systems. The NMODL DSL automatically includes a number of generic pragmas which

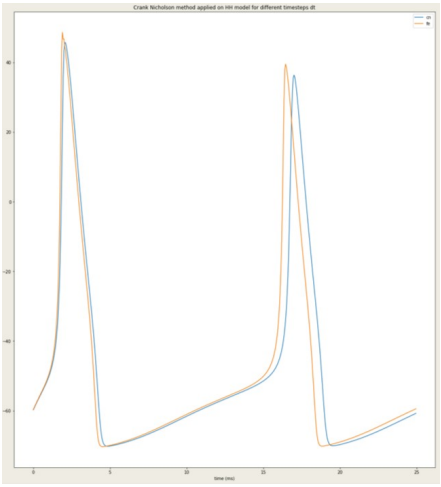

Figure 8: Comparison of the voltage evolution of a single compartment with respect to time when using implicit (orange) and explicit (blue) solvers: The response of the explicit solver is not yet matching the one of the implicit solver.

are replaced by platform specific pragmas at compile time based on the targeted architecture. An example of the results obtained on NVIDIA GPUs were presented at an NVIDIA GTC Conference [4].

Further developments are currently being investigated to further improve the existing source to source compiler. It includes refactoring of the software itself to perform optimizations before the code is actually presented to the backend compiler. No results are however yet ready to be presented as part of this report.

5 Conclusions

After the careful evaluation of Intel KNL hardware and the performance of coreNEURON software when the system was booted in either cache or flat mode, we have concluded that the benefits of the heavy code refactoring required by the flat mode were not substantial enough. We therefore focused on making the best use of the cache mode of the Intel KNL through the careful analysis and tuning of the software setup while spending more time on developments and enhancements of new functionalities needed by the scientific community. In addition to the developments and setup optimization on KNL-based system and the kernel optimizations previously performed on Intel KNC system, a number of proof of concepts such as in-memory compression, the implementation of an explicit solver as well as the study of the impact of the Spike Exchange on application scaling have been investigated to anticipate future issues of the software towards Exascale computing. Those developments and know how have greatly contributed to support the work performed as part of the on-going INCITE grant which aims at studying brain plasticity. It will allow BBP to be fully equipped to move its INCITE workload from Blue Gene/Q to KNL Theta over the years to come.

References

[1] D. Bayly. In-memory compression for Neuroscience Applications. *Google Summer of Code*, 2017.

[2] M. L. Hines et al. Fully implicit simulations of single neurons. *Journal of Computational Neuroscience*, 25(3):439–448, 2008.

[3] M. L. Hines et al. Comparison of neuronal spike exchange methods on a Blue Gene/P supercomputer. *Frontiers in Computational Neuroscience*, 18(5), November 2011.

[4] P. Kumbhar et al. Coreneuron: Morphologically detailed neuron simulations building, simulating and optimizing large neuron networks on gpus. NVIDIA GTC Conference, 2016.

[5] Timothée Ewart, Judit Planas, Francesco Cremonesi, Kai Langen, Felix Schürmann, and Fabien Delalondre. Neuromapp: A mini-application framework to improve neural simulators. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, pages 181–198, Cham, 2017. Springer International Publishing.

[6] Pramod Kumbhar, Michael Hines, Aleksandr Ovcharenko, Damian A. Mallon, James King, Florentino Sainz, Felix Schürmann, and Fabien Delalondre. Leveraging a cluster-booster architecture for brain-scale simulations. In Julian M. Kunkel, Pavan Balaji, and Jack Dongarra, editors, *High Performance Computing*, pages 363–380, Cham, 2016. Springer International Publishing.

Annex 1 – Plasticity use case

Use Case Description

The term "synaptic plasticity" refers to the capability of neurons to create, adapt or eliminate synaptic connections with other neurons. Synaptic plasticity is a ubiquitous phenomenon in the mammalian brain and it is considered to be the biological substrate of learning and memory. However, the exact mechanisms behind synaptic plasticity are still largely unknown, mainly due to the current limitations of experimental procedures.

Many theoretical models have been proposed to describe bits of the accumulating experimental evidence. However, those models often target only a specific phenomenology (i.e. spike-timing dependent plasticity, homeostatic synaptic scaling or structural plasticity) and so fail to explain how the underlying plastic mechanisms work and how the different mechanisms are orchestrated together in the brain. A unifying model of the mechanisms driving synaptic plasticity is clearly lacking and it is very unlikely that simply combining several phenomenological models is going to tell us anything about the connection between synaptic plasticity and learning or memory.

For obvious reasons, synaptic plasticity models need to be combined with neuron and connectivity models in order to study network effects. The most widely adopted approach is to rely on point-neuron models, connected in sparse random networks, neglecting important aspects of neuronal biophysics for simplicity and tractability. Although these simplifications are usually tolerable and even beneficial when modeling synaptic plasticity as in simple in vitro experiments, they pose a serious threat to the validity of the conclusions when the results are generalized as brain principles. It is indeed well known that the spatial extent of neurons and their connectivity have several important effects on synaptic plasticity, both directly and indirectly through network dynamics:

- The location of synapses on neuronal morphologies determines the outcome of typical LTP/LTD inducing protocols, see (Froemke, Poo, & Dan, 2005)
- Concurrent activation of strategically positioned synapses can regulate the plastic changes at other dendritic locations, i.e. apical trunk and tuft dendrites in layer 5 pyramidal cells, see (Sjöström & Häusser, 2006)
- Synchronous activation of neighbor synapses reduce the risk of synapse pruning in vivo (ref missing)

For these reasons, biologically detailed simulations of neural circuits are the only viable alternative nowadays to study how a local phenomenon such as synaptic plasticity could give rise to global learning rules and support information storage in the brain.

A model of the whole rat somatosensory cortex is an excellent system to study synaptic plasticity at network level. From a technical point of view, the main challenge of such a project is not simulating many neurons, but rather many synapses for a very long time. Furthermore, the synaptic model needs to be enriched to support long-term dynamics, such as calcium-based potentiation and depression, and so it is computationally more expensive than the current one.

Requirements

Circuit Size: Somatosensory cortex SI, one hemisphere only (roughly 5 million neurons, 8 billion functional synapses, 80 billion potential synapses[1]), see Figure 1.

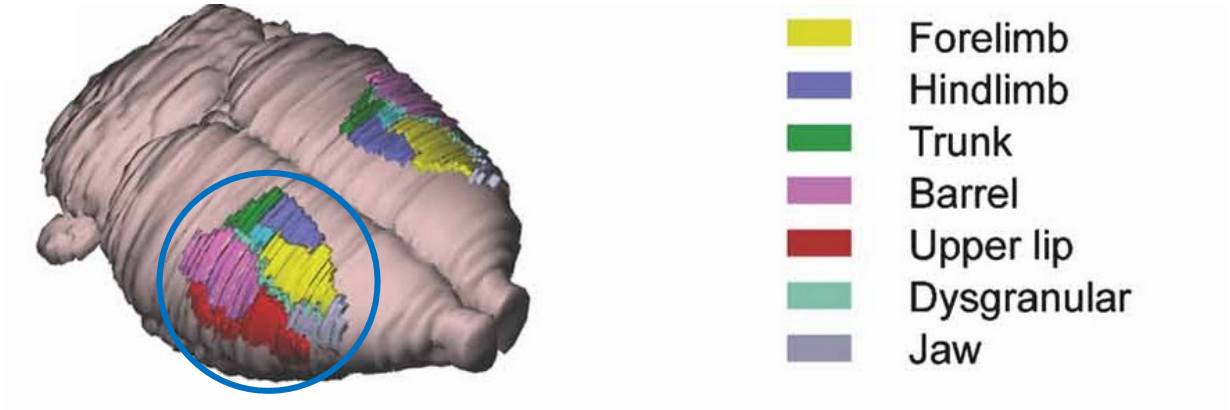

Figure 1: 3d reconstruction of somatosensory cortex SI. From (Hjornevik, et al., 2007).

Synapse Models: Two state process. Bidirectional transitions from potential synapse (simple linear system) to fully functional synapse (complex nonlinear system), see Mathematical Appendix. The model might change later. In particular, we never tested the rewiring dynamics and the proposed models are mostly placeholders. The most substantial change might be the need of better characterizing bouton creation and elimination to preserve our connectivity constraints. This would require the introduction of the axon in the simulation or a similar mean to explicitly describe the axonal dynamics. Despite this major change, we expect only modest differences in the model equations. Furthermore, the general idea of the synapse as a multi-state process with simple/complex states should hold.

Biological Time: minimum 1 day, maximum 30 days.

- *1 to 5 days* – The time frame should be long enough to experiment with basic learning tasks. We might expect a small fraction of potential synapses transitioning into functional and vice versa, while preserving the overall ratio between the two populations. Depending on the experiment, the turnover of functional synapses should vary between 10 % and 20 %. Given the time scale of the components of the functional synaptic model, we should be able to observe how neural activity and plasticity mechanisms shaped the actual values of all the synaptic variables. Again, any change should eventually be compensated by another one to preserve circuit stability.
- *5 to 30 days* – In such a long time frame we might expect major restructuring of the circuit. A substantial fraction of potential synapses will become functional and vice versa, while preserving the overall ratio between the two populations. Depending on the experiment, the turnover of functional synapses should vary between 20 % and 60%. In this setup we will probably study only the stability of microcircuit rewiring.

[1] A potential synapse is an apposition between two neurons that could host a synapse, although is not. Those appositions can be dynamically converted into functional synapses by plastic mechanisms.

Bibliography

Froemke, R. C., Poo, M.-m., & Dan, Y. (2005). Spike-timing-dependent synaptic plasticity depends on dendritic location. *Nature, 434*(7030), 221-225.

Hjornevik, T., Leergaard, T. B., Darine, D., Moldestad, O., Dale, A. M., Willoch, F., & Bjaalie, J. G. (2007). Three-dimensional atlas system for mouse and rat brain imaging data. *Frontiers in Neuroinformatics, 1*(00004). doi:10.3389/neuro.11.004.2007

Sjöström, P. J., & Häusser, M. (2006). A cooperative switch determines the sign of synaptic plasticity in distal dendrites of neocortical pyramidal neurons. *Neuron, 51*(2), 227- 238.

Argonne Leadership Computing Facility
Argonne National Laboratory
9700 South Cass Avenue, Bldg. #240
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC